SAM4 Pulse Width Modulation Controller (PWM)

This driver for SAM4E and SAM4S devices provides an interface for the configuration and management of the device’s Pulse Width Modulation functionality. The Pulse Width Modulation Controller has four independently controllable channels. Each channel controls two complementary square-wave outputs. The characteristics of the output waveform such as period, duty-cycle, polarity, and dead-times (also called dead-bands or non-overlapping times) are also configurable.

The outline of this documentation is as follows:

- Prerequisites
- Module Overview
- Special Considerations
- Extra Information
- Examples
- API Overview
Table of Contents

SAM4 Pulse Width Modulation Controller (PWM) ........................................ 1
Software License ............................................................................................. 5
1. Prerequisites ................................................................................................. 6
2. Module Overview ............................................................................................ 7
3. Special Considerations ................................................................................... 8
   3.1. I/O Lines .................................................................................................. 8
   3.2. Power Management ................................................................................. 8
   3.3. Interrupt Sources .................................................................................... 8
4. Extra Information .......................................................................................... 9
5. Examples ......................................................................................................... 10
6. API Overview .................................................................................................. 11
   6.1. Variable and Type Definitions ............................................................... 11
       6.1.1. Type pwm_ch_t ................................................................................ 11
   6.2. Structure Definitions .............................................................................. 11
       6.2.1. Struct pwm_channel_t .................................................................... 11
       6.2.2. Struct pwm_clock_t ....................................................................... 12
       6.2.3. Struct pwm_cmp_t .......................................................................... 12
       6.2.4. Struct pwm_fault_t ......................................................................... 13
       6.2.5. Struct pwm_output_t .................................................................... 13
       6.2.6. Struct pwm_protect_t ..................................................................... 13
   6.3. Macro Definitions ................................................................................... 14
       6.3.1. Macro PWM_INVALID_ARGUMENT ........................................... 14
   6.4. Function Definitions ............................................................................... 14
       6.4.1. Function pwm_channel_disable() ............................................... 14
       6.4.2. Function pwm_channel_disable_interrupt() ............................... 14
       6.4.3. Function pwm_channel_enable() ................................................. 15
       6.4.4. Function pwm_channel_enable_interrupt() ................................ 15
       6.4.5. Function pwm_channel_get_counter() ....................................... 15
       6.4.6. Function pwm_channel_get_interrupt_mask() ............................ 16
       6.4.7. Function pwm_channel_get_interrupt_status() ............................ 16
       6.4.8. Function pwm_channel_get_status() .......................................... 17
       6.4.9. Function pwm_channel_init() ..................................................... 17
       6.4.10. Function pwm_channel_update_additional_edge() ..................... 17
       6.4.11. Function pwm_channel_update_dead_time() ............................... 18
       6.4.12. Function pwm_channel_update_duty() ....................................... 18
       6.4.13. Function pwm_channel_update_output() ................................... 19
       6.4.14. Function pwm_channel_update_period() ................................... 19
       6.4.15. Function pwm_channel_update_polarity_mode() ........................ 20
       6.4.16. Function pwm_channel_update_spread() ................................... 20
       6.4.17. Function pwm_cmp_change_setting() ......................................... 21
       6.4.18. Function pwm_cmp_disable_interrupt() .................................... 21
       6.4.19. Function pwm_cmp_enable_interrupt() ....................................... 21
       6.4.20. Function pwm_cmp_get_period_counter() ................................. 22
       6.4.21. Function pwm_cmp_get_update_counter() ................................. 22
       6.4.22. Function pwm_cmp_init() ............................................................ 23
       6.4.23. Function pwm_disable_protect() ................................................ 23
       6.4.24. Function pwm_enable_protect() .................................................. 23
       6.4.25. Function pwm_fault_clear_status() ............................................ 24
       6.4.26. Function pwm_fault_get_input_level() ....................................... 24
       6.4.27. Function pwm_fault_get_status() ............................................... 24
       6.4.28. Function pwm_fault_init() ............................................................ 25
6.4.29. Function pwm_get_interrupt_mask() ............................ 25
6.4.30. Function pwm_get_interrupt_status() ....................... 25
6.4.31. Function pwm_get_protect_status() ............................ 26
6.4.32. Function pwm_init() ................................................. 26
6.4.33. Function pwm_pdc_disable_interrupt() ....................... 27
6.4.34. Function pwm_pdc_enable_interrupt() ......................... 27
6.4.35. Function pwm_pdc_set_request_mode() ....................... 28
6.4.36. Function pwm_stepper_motor_init() ............................ 28
6.4.37. Function pwm_sync_change_period() ......................... 28
6.4.38. Function pwm_sync_disable_interrupt() ...................... 29
6.4.39. Function pwm_sync_enable_interrupt() ....................... 29
6.4.40. Function pwm_sync_get_period_counter() .................... 30
6.4.41. Function pwm_sync_init() ........................................... 30
6.4.42. Function pwm_sync_unlock_update() .......................... 31

6.5. Enumeration Definitions .................................................. 31
6.5.1. Enum _pwm_ch_t ...................................................... 31
6.5.2. Enum pmc_cmp_unit_t .............................................. 31
6.5.3. Enum pwm_additional_edge_mode_t ............................. 32
6.5.4. Enum pwm_align_t ................................................... 32
6.5.5. Enum pwm_cmp_interrupt_t ...................................... 32
6.5.6. Enum pwm_counter_event_t ....................................... 32
6.5.7. Enum pwm_fault_id_t ............................................... 33
6.5.8. Enum pwm_level_t .................................................. 33
6.5.9. Enum pwm_pdc_interrupt_t ....................................... 33
6.5.10. Enum pwm_pdc_request_mode_t ................................. 33
6.5.11. Enum pwm_protect_reg_group_t ................................. 33
6.5.12. Enum pwm_spread_spectrum_mode_t ......................... 34
6.5.13. Enum pwm_stepper_motor_pair_t ............................... 34
6.5.14. Enum pwm_sync_interrupt_t ..................................... 34
6.5.15. Enum pwm_sync_update_mode_t ................................. 34

7. Extra Information for Pulse Width Modulation Controller ......... 36
7.1. Acronyms ................................................................. 36
7.2. Dependencies ........................................................... 36
7.3. Errata ..................................................................... 36
7.4. Module History ......................................................... 36

8. Examples for SAM Pulse Width Modulation Controller
(PWM) .................................................................................... 37
8.1. Quick Start guide for SAM PWM module ............................ 37
8.1.1. Basic Use Case ........................................................ 37
8.1.2. Setup Steps ........................................................... 37
8.1.3. Usage Steps ........................................................... 39
8.1.4. Advanced Use Case ............................................... 39
8.1.5. Setup Steps ........................................................... 40
8.1.6. Usage Steps ........................................................... 42
8.2. Pulse Width Modulator Controller - Example controlling an LED ............... 42
8.2.1. Purpose ................................................................. 42
8.2.2. Requirements ........................................................ 43
8.2.3. Main Files ............................................................. 43
8.2.4. Compilation Information ........................................ 43
8.2.5. Usage ................................................................. 43
8.3. Pulse Width Modulator Controller - Example synchronous channel
LED control ............................................................................ 44
8.3.1. Purpose ................................................................. 44
8.3.2. Requirements ........................................................ 44
8.3.3. Main Files ............................................................. 44
8.3.4. Compilation Information ........................................ 44
8.3.5. Usage ................................................................. 44

Index ....................................................................................... 46
Software License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.
4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
1. **Prerequisites**

There are no prerequisites for this module.
2. **Module Overview**

The PWM Controller has four independently controllable channels. Each channel controls two complementary square-wave outputs. The characteristics of the output waveform such as period, duty-cycle, polarity, and dead-times (also called dead-bands or non-overlapping times) are also configurable.

All PWM Controller channels integrate a double-buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum, the duty-cycle, the additional edge register, or the dead-times.

PWM channels can be linked together as synchronous channels, in order to be able to update their duty-cycle or dead-times at the same time. Synchronous channel duty cycle update can be performed by via a Peripheral DMA Controller (PDC) channel, which offers buffer transfer without processor intervention.

---

**Note**

The SAM4E PWM Controller includes a spread-spectrum counter to allow a constantly varying period (only for Channel 0). This counter may be useful in minimizing electromagnetic interference or reducing the acoustic noise of a PWM driven motor.

The PWM Controller provides eight independent comparison units, each capable of comparing a programmed value to the counter of the synchronous channels (counter of channel 0). These comparisons can be used to generate software interrupts, to trigger pulses on the two independent event lines (in order to synchronize ADC conversions with a lot of flexibility independently of the PWM outputs) and to trigger PDC transfer requests.

The PWM block provides a fault-protection mechanism with eight fault inputs, capable of detecting fault conditions and overriding the PWM outputs asynchronously (outputs forced to 0, 1, or high impedance).
3. Special Considerations

3.1 I/O Lines

The pins used for interfacing to the PWM are multiplexed with GPIO lines. The user application must first program the GPIO controller, in order to assign the desired PWM pins to their peripheral function. If any PWM I/O lines are not used by the target application, they can be used for other purposes by the GPIO controller.

3.2 Power Management

The PWM is not continuously clocked. The user application must first enable the PWM clock in the Power Management Controller (PMC) before using the PWM. Also, if the application does not require PWM operations, the PWM clock can be stopped whenever it is not needed and can be restarted at a later time. In this case, the PWM will resume its operations from where it left off.

3.3 Interrupt Sources

The PWM interrupt line is connected to one of the internal sources of the Nested Vectored Interrupt Controller (NVIC). Using the PWM interrupt requires that the NVIC be configured first.

Note

It is recommended that the PWM interrupt line not be used in edge sensitive mode.
4. **Extra Information**

For extra information, see *Extra Information for Pulse Width Modulation Controller*. This includes:

- Acronyms
- Dependencies
- Errata
- Module History
5. **Examples**

For a list of examples related to this driver, see Examples for SAM Pulse Width Modulation Controller (PWM).
6. API Overview

6.1 Variable and Type Definitions

6.1.1 Type pwm_ch_t

```c
typedef enum _pwm_ch_t pwm_ch_t
```

PWM channel numbers.

6.2 Structure Definitions

6.2.1 Struct pwm_channel_t

PWM channel mode input parameter configuration.

Table 6-1. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_additional_edge_mode_t</td>
<td>additional_edge_mode</td>
<td>Additional Edge Mode (SAM4E devices only).</td>
</tr>
<tr>
<td>pwm_align_t</td>
<td>alignment</td>
<td>Channel alignment.</td>
</tr>
<tr>
<td>bool</td>
<td>b_deadtime_generator</td>
<td>Enable/disable channel dead-time generator (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>bool</td>
<td>b_pwmh_output_inverted</td>
<td>Enable/disable channel dead-time PWMH output inversion (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>bool</td>
<td>b_pwml_output_inverted</td>
<td>Enable/disable channel dead-time PWML output inversion (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>bool</td>
<td>b_sync_ch</td>
<td>Enable/disable Synchronous Channel (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>uint32_t</td>
<td>channel</td>
<td>Channel number.</td>
</tr>
<tr>
<td>pwm_counter_event_t</td>
<td>counter_event</td>
<td>Channel counter event (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>pwm_fault_id_t</td>
<td>fault_id</td>
<td>Channel fault ID (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).</td>
</tr>
<tr>
<td>pwm_output_t</td>
<td>output_selection</td>
<td>Channel output.</td>
</tr>
<tr>
<td>pwm_level_t</td>
<td>polarity</td>
<td>Channel initial polarity.</td>
</tr>
<tr>
<td>pwm_spread_spectrum_mode_t</td>
<td>spread_spectrum_mode</td>
<td>Spread spectrum mode (SAM4E devices only).</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_additional_edge</td>
<td>Additional edge value (range 0 to 65535) (SAM4E devices only).</td>
</tr>
</tbody>
</table>
### 6.2.2 Struct pwm_clock_t

Input parameters when initializing PWM.

**Table 6-2. Members**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>ul_clka</td>
<td>Frequency of clock A in Hz (set 0 to switch it off).</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_clkb</td>
<td>Frequency of clock B in Hz (set 0 to switch it off).</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_mck</td>
<td>Frequency of master clock in Hz.</td>
</tr>
</tbody>
</table>

### 6.2.3 Struct pwm_cmp_t

PWM comparison configuration structure.

**Table 6-3. Members**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>b_enable</td>
<td>Enable/disable comparison unit.</td>
</tr>
<tr>
<td>bool</td>
<td>b_is_decrementing</td>
<td>Comparison mode.</td>
</tr>
<tr>
<td>bool</td>
<td>b_pulse_on_line_0</td>
<td>Enable/disable the match pulse event generation on PWM line 0.</td>
</tr>
</tbody>
</table>
### Table 6-4. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>b_pulse_on_line_1</td>
<td>Enable/disable the match pulse event generation on PWM line 1.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_period</td>
<td>Comparison period value.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_trigger</td>
<td>Comparison trigger value.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_update_period</td>
<td>Comparison update period value.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_value</td>
<td>Comparison value.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>unit</td>
<td>Comparison unit number.</td>
</tr>
</tbody>
</table>

### 6.2.4 Struct pwm_fault_t

PWM fault input behavior configuration.

### Table 6-5. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>b_clear</td>
<td>Enable/disable fault flag clearing.</td>
</tr>
<tr>
<td>bool</td>
<td>b_filtered</td>
<td>Enable/disable fault filtering.</td>
</tr>
<tr>
<td>pwm_fault_id_t</td>
<td>fault_id</td>
<td>Fault ID.</td>
</tr>
<tr>
<td>pwm_level_t</td>
<td>polarity</td>
<td>Polarity of fault input.</td>
</tr>
</tbody>
</table>

### 6.2.5 Struct pwm_output_t

PWM channel output configuration.

### Table 6-6. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>b_override_pwmh</td>
<td>Enable/disable the PWMH output override.</td>
</tr>
<tr>
<td>bool</td>
<td>b_override_pwml</td>
<td>Enable/disable the PWML output override.</td>
</tr>
<tr>
<td>pwm_level_t</td>
<td>override_level_pwmh</td>
<td>The override output level for PWMH.</td>
</tr>
<tr>
<td>pwm_level_t</td>
<td>override_level_pwml</td>
<td>The override output level for PWML.</td>
</tr>
</tbody>
</table>

### 6.2.6 Struct pwm_protect_t

PWM write-protect information configuration.

### Table 6-7. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>ul_hw_status</td>
<td>Bitmask of the PWM register groups for write-protect hardware status.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>ul_offset</td>
<td>Offset address of the PWM register to which a write access has been attempted.</td>
</tr>
</tbody>
</table>
6.3 Macro Definitions

6.3.1 Macro PWM_INVALID_ARGUMENT

#define PWM_INVALID_ARGUMENT

Invalid argument error.

6.4 Function Definitions

6.4.1 Function pwm_channel_disable()

Disable the specified PWM channel.

```c
void pwm_channel_disable(  
Pwm * p_pwm,  
uint32_t ul_channel)
```

**Note**

A disabled PWM channel can be re-initialized using `pwm_channel_init()`.

**Table 6-7. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_channel</td>
<td>PWM channel number</td>
</tr>
</tbody>
</table>

6.4.2 Function pwm_channel_disable_interrupt()

Disable a PWM channel's counter event and fault protection interrupt.

```c
void pwm_channel_disable_interrupt(  
Pwm * p_pwm,  
uint32_t ul_event,  
uint32_t ul_fault)
```

**Table 6-8. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_event</td>
<td>Channel number on which the counter event interrupt should be disabled.</td>
</tr>
</tbody>
</table>
### 6.4.3 Function `pwm_channel_enable()`

Enable the specified PWM channel.

```c
void pwm_channel_enable(
    Pwm * p_pwm,
    uint32_t ul_channel)
```

**Note**
The PWM channel should be initialized by `pwm_channel_init()` before it is enabled.

### Table 6-9. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>ul_fault</td>
<td>Channel number on which the fault protection interrupt should be disabled (the parameter value is ignored by SAM3N/SAM4N/SAM4C devices).</td>
</tr>
</tbody>
</table>

### 6.4.4 Function `pwm_channel_enable_interrupt()`

Enable a PWM channel's counter event and fault protection interrupt.

```c
void pwm_channel_enable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_event,
    uint32_t ul_fault)
```

### Table 6-10. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_event</td>
<td>Channel number on which the counter event interrupt should be enabled.</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_fault</td>
<td>Channel number on which the fault protection interrupt should be enabled (the parameter value is ignored by SAM3N/SAM4N/SAM4C devices).</td>
</tr>
</tbody>
</table>

### 6.4.5 Function `pwm_channel_get_counter()`

Get a PWM channel counter value.
**uint32_t pwm_channel_get_counter(Pwm * p_pwm, pwm_channel_t * p_channel)**

Table 6-11. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_channel</td>
<td>Channel configuration structure pointer</td>
</tr>
</tbody>
</table>

**Returns**
The PWM channel counter value.

**6.4.6 Function pwm_channel_get_interrupt_mask()**

*Get the PWM channel counter event and fault protection trigger interrupt mask.*

**uint32_t pwm_channel_get_interrupt_mask(Pwm * p_pwm)**

Table 6-12. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**
The PWM channel counter event and fault protection trigger interrupt mask. Refer to the section called "pwm interrupt mask register" in the device-specific datasheet for more information.

**6.4.7 Function pwm_channel_get_interrupt_status()**

*Get the PWM channel counter event, and fault protection trigger interrupt status.*

**uint32_t pwm_channel_get_interrupt_status(Pwm * p_pwm)**

Table 6-13. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**
The channel counter event, and fault protection trigger interrupt status. Refer to the section called "PWM Interrupt Status Register" in the device-specific datasheet for more information.
6.4.8 Function pwm_channel_get_status()

Check which PWM channel(s) are enabled.

```c
uint32_t pwm_channel_get_status(
    Pwm * p_pwm)
```

Table 6-14. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

Returns

The bitmask of enabled PWM channel(s). Refer to the section called "PWM Status Register" in the device-specific datasheet for further information.

6.4.9 Function pwm_channel_init()

Initialize a PWM channel.

```c
uint32_t pwm_channel_init(
    Pwm * p_pwm,
    pwm_channel_t * p_channel)
```

Table 6-15. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_channel</td>
<td>Channel configuration structure pointer</td>
</tr>
</tbody>
</table>

Returns

The PWM channel initialization status.

Table 6-16. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PWM channel was initialized correctly</td>
</tr>
</tbody>
</table>

6.4.10 Function pwm_channel_update_additional_edge()

Change a PWM channel's additional edge value and mode.

```c
void pwm_channel_update_additional_edge(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    uint32_t ul_additional_edge,
    pwm_additional_edge_mode_t additional_edge_mode)
6.4.11 Function pwm_channel_update_dead_time()

Change a PWM channel's dead-time.

```c
void pwm_channel_update_dead_time(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    uint16_t us_deadtime_pwmh,
    uint16_t us_deadtime_pwml)
```

6.4.12 Function pwm_channel_update_duty()

Set the duty cycle of a PWM channel.

```c
uint32_t pwm_channel_update_duty(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    uint32_t ul_duty)
```
Returns

An indication of whether the PWM channel duty cycle was successfully changed.

Table 6-20. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PWM channel duty cycle was successfully changed</td>
</tr>
<tr>
<td>PWM_INVALID_ARGUMENT</td>
<td>Invalid parameter(s) in the channel configuration</td>
</tr>
</tbody>
</table>

6.4.13 Function pwm_channel_update_output()

Change a PWM channel output selection.

```c
void pwm_channel_update_output(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    pwm_output_t * p_output,
    bool b_sync)
```

Table 6-21. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[out]</td>
<td>p_channel</td>
<td>Channel configuration structure pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_output</td>
<td>PWM channel output selection</td>
</tr>
<tr>
<td>[in]</td>
<td>b_sync</td>
<td>Set to true to change the output synchronously (at the beginning of the next PWM period). Set to false to change the output asynchronously (at the end of the execution of the function).</td>
</tr>
</tbody>
</table>

6.4.14 Function pwm_channel_update_period()

Set the period of a PWM channel.

```c
uint32_t pwm_channel_update_period(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    uint32_t ul_period)
```

Table 6-22. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in, out]</td>
<td>p_channel</td>
<td>Channel configuration structure pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_period</td>
<td>Period value</td>
</tr>
</tbody>
</table>
Returns

An indication of whether the PWM channel period was successfully changed.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PWM channel period was successfully changed</td>
</tr>
<tr>
<td>PWM_INVALID_ARGUMENT</td>
<td>Invalid parameter(s) in the channel configuration</td>
</tr>
</tbody>
</table>

### 6.4.15 Function pwm_channel_update_polarity_mode()

Change a PWM channel's polarity mode.

```c
void pwm_channel_update_polarity_mode(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    bool polarity_inversion_flag,
    pwm_level_t polarity_value)
```

**Note**

This function is only available on SAM4E devices.

### 6.4.16 Function pwm_channel_update_spread()

Set a PWM channel's spread spectrum value.

```c
void pwm_channel_update_spread(
    Pwm * p_pwm,
    pwm_channel_t * p_channel,
    uint32_t ul_spread)
```

**Note**

This function is only available on SAM4E devices.

### Table 6-24. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in, out]</td>
<td>p_channel</td>
<td>Channel configuration structure pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>polarity_inversion_flag</td>
<td>Polarity inversion (true for inversion, false otherwise)</td>
</tr>
<tr>
<td>[in]</td>
<td>polarity_value</td>
<td>Polarity value</td>
</tr>
</tbody>
</table>

### Table 6-25. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>
### 6.4.17 Function `pwm_cmp_change_setting()`

*Change the settings for a PWM comparison unit.*

```c
uint32_t pwm_cmp_change_setting(
    Pwm * p_pwm,
    pwm_cmp_t * p_cmp)
```

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_cmp</td>
<td>Comparison unit configuration structure</td>
</tr>
</tbody>
</table>

**Returns**
The PWM comparison unit configuration change status.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PWM comparison unit was configured correctly</td>
</tr>
</tbody>
</table>

### 6.4.18 Function `pwm_cmp_disable_interrupt()`

*Disable the specified PWM comparison unit interrupt.*

```c
void pwm_cmp_disable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources,
    pwm_cmp_interrupt_t type)
```

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Comparison unit number</td>
</tr>
<tr>
<td>[in]</td>
<td>type</td>
<td>Select a match or update interrupt</td>
</tr>
</tbody>
</table>

### 6.4.19 Function `pwm_cmp_enable_interrupt()`

*Enable the specified PWM comparison unit interrupt.*
void pwm_cmp_enable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources,
    pwm_cmp_interrupt_t type)

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Comparison unit number</td>
</tr>
<tr>
<td>[in]</td>
<td>type</td>
<td>Select whether to match or update interrupts</td>
</tr>
</tbody>
</table>

### 6.4.20 Function pwm_cmp_get_period_counter()

_Get the specified PWM comparison unit period counter._

```c
uint32_t pwm_cmp_get_period_counter(
    Pwm * p_pwm,
    uint32_t ul_cmp_unit)
```

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_cmp_unit</td>
<td>PWM comparison unit number.</td>
</tr>
</tbody>
</table>

**Returns**
The PWM comparison unit period counter.

### 6.4.21 Function pwm_cmp_get_update_counter()

_Get the specified PWM comparison unit update period counter._

```c
uint32_t pwm_cmp_get_update_counter(
    Pwm * p_pwm,
    uint32_t ul_cmp_unit)
```

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_cmp_unit</td>
<td>PWM comparison unit number.</td>
</tr>
</tbody>
</table>

**Returns**
The PWM comparison unit update period counter.
6.4.22  Function pwm_cmp_init()

*Initialize a PWM comparison unit.*

```c
uint32_t pwm_cmp_init(
    Pwm * p_pwm,
    pwm_cmp_t * p_cmp)
```

**Table 6-32. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_cmp</td>
<td>Comparison unit configuration structure</td>
</tr>
</tbody>
</table>

**Returns**

The PWM comparison unit initialization status.

**Table 6-33. Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PWM comparison unit was initialized correctly</td>
</tr>
</tbody>
</table>

6.4.23  Function pwm_disable_protect()

*Disable the write protection of the PWM registers.*

```c
void pwm_disable_protect(
    Pwm * p_pwm,
    uint32_t ul_group)
```

**Note**

Only a hardware reset of the PWM controller (handled by PMC) can disable hardware write protection.

**Table 6-34. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_group</td>
<td>Bitmask of PWM register group</td>
</tr>
</tbody>
</table>

6.4.24  Function pwm_enable_protect()

*Enable the PWM registers write protect.*

```c
void pwm_enable_protect(
    Pwm * p_pwm,
    uint32_t ul_group,
    bool b_sw)
```
Table 6-35. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_group</td>
<td>Bitmask of PWM register group</td>
</tr>
<tr>
<td>[in]</td>
<td>b_sw</td>
<td>Protection, true for software protection and false for hardware protection.</td>
</tr>
</tbody>
</table>

6.4.25 Function pwm_fault_clear_status()

Clear a PWM fault input.

```c
void pwm_fault_clear_status(
    Pwm * p_pwm,
    pwm_fault_id_t id)
```

6.4.26 Function pwm_fault_get_input_level()

Get the PWM fault input level.

```c
pwm_level_t pwm_fault_get_input_level(
    Pwm * p_pwm,
    pwm_fault_id_t id)
```

Table 6-37. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>id</td>
<td>Fault ID</td>
</tr>
</tbody>
</table>

Returns

The PWM fault input Level.

6.4.27 Function pwm_fault_get_status()

Get the PWM fault status.

```c
uint32_t pwm_fault_get_status(
    Pwm * p_pwm)
```
Table 6-38. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**
The bitmask of IDs for all currently active PWM faults.

### 6.4.28 Function pwm_fault_init()

*Initialize the PWM fault input behavior.*

```c
uint32_t pwm_fault_init(
    Pwm * p_pwm,
    pwm_fault_t * p_fault)
```

Table 6-39. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>p_fault</td>
<td>Fault configuration structure pointer</td>
</tr>
</tbody>
</table>

**Returns**
The fault input initialization status.

Table 6-40. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Fault input behavior initialized correctly</td>
</tr>
</tbody>
</table>

### 6.4.29 Function pwm_get_interrupt_mask()

*Get the PDC transfer, synchronous channels and comparison interrupt mask.*

```c
uint32_t pwm_get_interrupt_mask(
    Pwm * p_pwm)
```

Table 6-41. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**
The PDC transfer, synchronous channels and comparison interrupt mask. Refer to the section called "PWM Interrupt Mask Register 2" in the device-specific datasheet.

### 6.4.30 Function pwm_get_interrupt_status()
Get the PDC transfer, synchronous channels and comparison interrupt status.

```c
uint32_t pwm_get_interrupt_status(
    Pwm * p_pwm)
```

**Table 6-42. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**

The PDC transfer, synchronous channels and comparison interrupt status. Refer to the section called "PWM Interrupt Status Register 2" in the device-specific datasheet.

6.4.31 Function `pwm_get_protect_status()`

Get the PWM write protection status.

```c
bool pwm_get_protect_status(
    Pwm * p_pwm,
    pwm_protect_t * p_protect)
```

**Table 6-43. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[out]</td>
<td>p_protect</td>
<td>Pointer to a PWM protect structure</td>
</tr>
</tbody>
</table>

**Returns**

The PWM write protection status.

**Table 6-44. Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Write protection is disabled</td>
</tr>
<tr>
<td>true</td>
<td>Write protection is enabled</td>
</tr>
</tbody>
</table>

6.4.32 Function `pwm_init()`

Initialize the PWM source clock (clock A and clock B).

```c
uint32_t pwm_init(
    Pwm * p_pwm,
    pwm_clock_t * clock_config)
```

**Table 6-45. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>
### Data direction

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_config</td>
<td>Pointer to a PWM clock configuration structure</td>
</tr>
</tbody>
</table>

**Returns**

The initialization result.

### Table 6-46. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initialization successful</td>
</tr>
<tr>
<td>PWM_INVALID_ARGUMENT</td>
<td>Invalid parameter(s) in the channel configuration</td>
</tr>
</tbody>
</table>

### 6.4.33 Function pwm_pdc_disable_interrupt()

*Disable a PDC transfer interrupt.*

```c
void pwm_pdc_disable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources)
```

### Table 6-47. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Bitmask of PWM PDC transfer interrupt sources</td>
</tr>
</tbody>
</table>

Where the input parameter `ul_sources` is a bitmask containing one or more of the following:

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_IDR2_ENDTX</td>
<td>Disable the PDC End of TX buffer interrupt</td>
</tr>
<tr>
<td>PWM_IDR2_TXBUFE</td>
<td>Disable the PDC TX buffer empty interrupt</td>
</tr>
</tbody>
</table>

### 6.4.34 Function pwm_pdc_enable_interrupt()

*Enable a PDC transfer interrupt.*

```c
void pwm_pdc_enable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources)
```

### Table 6-48. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Bitmask of PWM PDC transfer interrupt sources</td>
</tr>
</tbody>
</table>

Where the input parameter `ul_sources` is a bitmask containing one or more of the following:
### 6.4.35 Function `pwm_pdc_set_request_mode()`

Set PDC transfer request mode.

```c
void pwm_pdc_set_request_mode(
    Pwm * p_pwm,
    pwm_pdc_request_mode_t request_mode,
    uint32_t ul_cmp_unit)
```

**Note**

If the synchronous channels' update mode is `PWM_SYNC_UPDATE_MODE_0` on page 35 or `PWM_SYNC_UPDATE_MODE_1` on page 35, then input parameter `ul_pdc_request` will be ignored and the PDC transfer request will never occur.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_IER2_ENDTX</td>
<td>Enable the PDC End of TX buffer interrupt</td>
</tr>
<tr>
<td>PWM_IER2_TXBUFE</td>
<td>Enable the PDC TX buffer empty interrupt</td>
</tr>
</tbody>
</table>

### 6.4.36 Function `pwm_stepper_motor_init()`

Initialize the PWM stepper motor mode.

```c
void pwm_stepper_motor_init(
    Pwm * p_pwm,
    pwm_stepper_motor_pair_t pair,
    bool b_enable_gray,
    bool b_down)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>pair</td>
<td>The two PWM channels used by the stepper motor.</td>
</tr>
<tr>
<td>b_enable_gray</td>
<td>Enable/disable gray count</td>
</tr>
<tr>
<td>b_down</td>
<td>Counter direction, true to count down, false to count up</td>
</tr>
</tbody>
</table>

### 6.4.37 Function `pwm_sync_change_period()`

Set the time period between each update of the PWM synchronous channels.
```c
void pwm_sync_change_period(
    Pwm * p_pwm,
    uint32_t ul_update_period)
```

Table 6-51. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_update_period</td>
<td>Time between each update of the synchronous channels</td>
</tr>
</tbody>
</table>

6.4.38 Function pwm_sync_disable_interrupt()

Disable a synchronous channel interrupt.

```c
void pwm_sync_disable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources)
```

Table 6-52. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Bitmask of PWM synchronous channels' interrupt sources</td>
</tr>
</tbody>
</table>

Where the input parameter ul_sources is a bitmask containing one or more of the following:

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_IDR2_WRDY</td>
<td>Disable the synchronous channels' Write Ready for update interrupt</td>
</tr>
<tr>
<td>PWM_IDR2_UNRE</td>
<td>Disable the Synchronous channels' update underrun error interrupt</td>
</tr>
</tbody>
</table>

6.4.39 Function pwm_sync_enable_interrupt()

Enable a PWM synchronous channel interrupt.

```c
void pwm_sync_enable_interrupt(
    Pwm * p_pwm,
    uint32_t ul_sources)
```

Table 6-53. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_sources</td>
<td>Bitmask of PWM synchronous channel interrupt sources</td>
</tr>
</tbody>
</table>
Where the input parameter `ul_sources` is a bitmask containing one or more of the following:

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_IER2_WRDY</td>
<td>Enable the synchronous channels’ Write Ready for update interrupt</td>
</tr>
<tr>
<td>PWM_IER2_UNRE</td>
<td>Enable the Synchronous channels’ update underrun error interrupt</td>
</tr>
</tbody>
</table>

### 6.4.40 Function `pwm_sync_get_period_counter()`

*Get the PWM synchronization update period counter.*

```c
uint32_t pwm_sync_get_period_counter(
    Pwm * p_pwm)
```

**Table 6-54. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

**Returns**

The PWM synchronization update Period Counter.

### 6.4.41 Function `pwm_sync_init()`

*Initialize the PWM synchronous channels’ update mode and period.*

```c
uint32_t pwm_sync_init(
    Pwm * p_pwm,
    pwm_sync_update_mode_t mode,
    uint32_t ul_update_period)
```

**Table 6-55. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in, out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
<tr>
<td>[in]</td>
<td>mode</td>
<td>Synchronous channels update mode</td>
</tr>
<tr>
<td>[in]</td>
<td>ul_update_period</td>
<td>Time between each update of the synchronous channel</td>
</tr>
</tbody>
</table>

**Returns**

The PWM channel initialization status.

**Table 6-56. Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PWM channel was initialized correctly</td>
</tr>
</tbody>
</table>
6.4.42 Function pwm_sync_unlock_update()

Unlock the synchronous channels update.

```c
void pwm_sync_unlock_update(
    Pwm * p_pwm)
```

**Note**

After being unlocked the synchronous channels will be updated at the next PWM period.

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>p_pwm</td>
<td>Module hardware register base address pointer</td>
</tr>
</tbody>
</table>

6.5 Enumeration Definitions

### 6.5.1 Enum _pwm_ch_t

PWM channel numbers.

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_CHANNEL_0</td>
<td>PWM channel 0.</td>
</tr>
<tr>
<td>PWM_CHANNEL_1</td>
<td>PWM channel 1.</td>
</tr>
<tr>
<td>PWM_CHANNEL_2</td>
<td>PWM channel 2.</td>
</tr>
<tr>
<td>PWM_CHANNEL_3</td>
<td>PWM channel 3.</td>
</tr>
</tbody>
</table>

### 6.5.2 Enum pmc_cmp_unit_t

PWM comparison unit.

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_CMP_UNIT_0</td>
<td>PWM comparison unit 0.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_1</td>
<td>PWM comparison unit 1.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_2</td>
<td>PWM comparison unit 2.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_3</td>
<td>PWM comparison unit 3.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_4</td>
<td>PWM comparison unit 4.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_5</td>
<td>PWM comparison unit 5.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_6</td>
<td>PWM comparison unit 6.</td>
</tr>
<tr>
<td>PWM_CMP_UNIT_7</td>
<td>PWM comparison unit 7.</td>
</tr>
</tbody>
</table>
6.5.3 Enum pwm_additional_edge_mode_t

PWM additional edge (SAM4E devices only).

Table 6-60. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_ADDITIONAL_EDGE_MODE_INC</td>
<td>The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, and the counter of channel x is incrementing.</td>
</tr>
<tr>
<td>PWM_ADDITIONAL_EDGE_MODE_DEC</td>
<td>The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, and the counter of channel x is incrementing.</td>
</tr>
<tr>
<td>PWM_ADDITIONAL_EDGE_MODE_BOTH</td>
<td>The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, whether the counter is incrementing or not.</td>
</tr>
</tbody>
</table>

6.5.4 Enum pwm_align_t

PWM channel alignment.

Table 6-61. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_ALIGN_LEFT</td>
<td>The period is left aligned.</td>
</tr>
<tr>
<td>PWM_ALIGN_CENTER</td>
<td>The period is center aligned.</td>
</tr>
</tbody>
</table>

6.5.5 Enum pwm_cmp_interrupt_t

PWM comparison interrupt.

Table 6-62. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_CMP_MATCH</td>
<td>Comparison unit match interrupt.</td>
</tr>
<tr>
<td>PWM_CMP_UPDATE</td>
<td>Comparison unit update interrupt.</td>
</tr>
</tbody>
</table>

6.5.6 Enum pwm_counter_event_t

PWM event.

Table 6-63. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_EVENT_PERIOD_END</td>
<td>The channel counter event occurs at the end of the PWM period.</td>
</tr>
<tr>
<td>PWM_EVENT_PERIOD_HALF_END</td>
<td>The channel counter event occurs halfway through the PWM period.</td>
</tr>
</tbody>
</table>
### 6.5.7 Enum pwm_fault_id_t

PWM fault input ID.

Table 6-64. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_FAULT_PWMFI1</td>
<td>Main oscillator fault.</td>
</tr>
<tr>
<td>PWM_FAULT_MAINOSC</td>
<td>ADC fault.</td>
</tr>
<tr>
<td>PWM_FAULT_ADC</td>
<td>Analog Comparator fault.</td>
</tr>
<tr>
<td>PWM_FAULT_TIMER_0</td>
<td>Timer 0 fault.</td>
</tr>
<tr>
<td>PWM_FAULT_TIMER_1</td>
<td>Timer 1 fault.</td>
</tr>
</tbody>
</table>

### 6.5.8 Enum pwm_level_t

PWM level.

Table 6-65. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_LOW</td>
<td>Low level.</td>
</tr>
<tr>
<td>PWM_HIGH</td>
<td>High level.</td>
</tr>
<tr>
<td>PWM_HIGHZ</td>
<td>High Impedance (SAM4E only).</td>
</tr>
</tbody>
</table>

### 6.5.9 Enum pwm_pdc_interrupt_t

PWM PDC transfer interrupt.

Table 6-66. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_PDC_TX_END</td>
<td>PDC transmit end.</td>
</tr>
<tr>
<td>PWM_PDC_TX_EMPTY</td>
<td>PDC transmit buffer empty.</td>
</tr>
</tbody>
</table>

### 6.5.10 Enum pwm_pdc_request_mode_t

PWM PDC transfer request mode.

Table 6-67. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_PDC_UPDATE_PERIOD_ELAPSED</td>
<td>PDC transfer request is set as soon as the update period elapses.</td>
</tr>
<tr>
<td>PWM_PDC_COMPARISON_MATCH</td>
<td>PDC transfer request is set as soon as the selected comparison matches.</td>
</tr>
</tbody>
</table>

### 6.5.11 Enum pwm_protect_reg_group_t

PWM write-protect register groups.
Table 6-68. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_GROUP_CLOCK</td>
<td>The write protect clock register.</td>
</tr>
<tr>
<td>PWM_GROUP_DISABLE</td>
<td>The write protect disable register.</td>
</tr>
<tr>
<td>PWM_GROUP_MODE</td>
<td>The write protect mode registers.</td>
</tr>
<tr>
<td>PWM_GROUP_PERIOD</td>
<td>The write protect period registers.</td>
</tr>
<tr>
<td>PWM_GROUP_DEAD_TIME</td>
<td>The write protect read time registers.</td>
</tr>
<tr>
<td>PWM_GROUP_FAULT</td>
<td>The write protect fault registers.</td>
</tr>
</tbody>
</table>

6.5.12 Enum pwm_spread_spectrum_mode_t

PWM Spread-Spectrum mode (SAM4E devices only).

Table 6-69. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_SPREAD_SPECTRUM_MODE_TRIANGULAR</td>
<td>The spread-spectrum counter starts to count from -SPRD when the channel 0 is enabled and counts upwards at each PWM period. When it reaches +SPRD, it restarts counting from -SPRD again.</td>
</tr>
<tr>
<td>PWM_SPREAD_SPECTRUM_MODE_RANDOM</td>
<td>The spread-spectrum counter is loaded with a new random value at each PWM period. This random value is uniformly distributed, and is between -SPRD and +SPRD.</td>
</tr>
</tbody>
</table>

6.5.13 Enum pwm_stepper_motor_pair_t

PWM channels used by motor stepper.

Table 6-70. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_STEPPER_MOTOR_CH_0_1</td>
<td>Channels 0 and 1.</td>
</tr>
<tr>
<td>PWM_STEPPER_MOTOR_CH_2_3</td>
<td>Channels 2 and 3.</td>
</tr>
</tbody>
</table>

6.5.14 Enum pwm_sync_interrupt_t

PWM synchronous channels interrupt.

Table 6-71. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_SYNC_WRITE_READY</td>
<td>Write Ready for synchronous channels update.</td>
</tr>
<tr>
<td>PWM_SYNC_UNDERRUN</td>
<td>Synchronous channels Update Underrun Error.</td>
</tr>
</tbody>
</table>

6.5.15 Enum pwm_sync_update_mode_t

PWM synchronous channels update mode.
<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_SYNC_UPDATE_MODE_0</td>
<td>Manual write of double-buffer registers and manual update of synchronous channels.</td>
</tr>
<tr>
<td>PWM_SYNC_UPDATE_MODE_1</td>
<td>Manual write of double-buffer registers and automatic update of synchronous channel.</td>
</tr>
<tr>
<td>PWM_SYNC_UPDATE_MODE_2</td>
<td>Automatic write of duty-cycle update registers by the PDC and automatic update of synchronous channel.</td>
</tr>
</tbody>
</table>
7. Extra Information for Pulse Width Modulation Controller

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input Output</td>
</tr>
<tr>
<td>ID</td>
<td>Identity</td>
</tr>
<tr>
<td>NVIC</td>
<td>Nested Vectored Interrupt Controller</td>
</tr>
<tr>
<td>PDC</td>
<td>Peripheral DMA Controller</td>
</tr>
<tr>
<td>PMC</td>
<td>Power Manager Controller</td>
</tr>
<tr>
<td>PWMH</td>
<td>Pulse Width Modulation High</td>
</tr>
<tr>
<td>PWML</td>
<td>Pulse Width Modulation Low</td>
</tr>
<tr>
<td>QSG</td>
<td>Quick Start Guide</td>
</tr>
</tbody>
</table>

7.2 Dependencies

This driver has the following dependencies:

- None

7.3 Errata

There are no errata related to this driver.

7.4 Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

<table>
<thead>
<tr>
<th>Changelog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial document release</td>
</tr>
</tbody>
</table>
8. Examples for SAM Pulse Width Modulation Controller (PWM)

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4 Pulse Width Modulation Controller (PWM). QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that QSGs can be compiled as a standalone application or be added to the user application.

- Quick Start guide for SAM PWM module
- Pulse Width Modulator Controller - Example controlling an LED
- Pulse Width Modulator Controller - Example synchronous channel LED control

8.1 Quick Start guide for SAM PWM module

This is the quick start guide for the SAM4 Pulse Width Modulation Controller (PWM), with step-by-step instructions on how to configure and use the drivers in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, for example, the main application function.

- Basic Use Case
- Advanced Use Case

8.1.1 Basic Use Case

In this basic use case, the PWM Controller module is configured as follows:

- Output a square-wave on PWM channel 0
- The frequency of the square-wave is 1kHz with a 50% duty cycle
- Clock A is used as the source clock
- The output wave can be verified on the device's selected output pin

8.1.2 Setup Steps

8.1.2.1 Prerequisites

- Power Management Controller (PMC) driver
- General Purpose I/O (GPIO) driver

8.1.2.2 Example Code

Add the following PWM initialization code segments to the beginning of your main application function:

```c
// PWM frequency in Hz.
#define PWM_FREQUENCY 1000

// Period value of PWM output waveform.
#define PERIOD_VALUE 100
```

```c
// Initial duty cycle value.
#define INIT_DUTY_VALUE 50
```
pwm_channel_t g_pwm_channel_led;

pmc_enable_periph_clk(ID_PWM);

pwm_channel_disable(PWM, PWM_CHANNEL_0);

// Set PWM clock A as PWM_FREQUENCY*PERIOD_VALUE (clock B is not used).
pwm_clock_t clock_setting = {
    .ul_clka = PWM_FREQUENCY * PERIOD_VALUE,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);

g_pwm_channel_led.channel = PWM_CHANNEL_0;

// Period is left-aligned.
g_pwm_channel_led.alignment = PWM_ALIGN_LEFT;
// Output waveform starts at a low level.
g_pwm_channel_led.polarity = PWM_LOW;
// Use PWM clock A as source clock.
g_pwm_channel_led.ul_prescaler = PWM_CMR_CPRE_CLKA;
// Period value of output waveform.
g_pwm_channel_led.ul_period = PERIOD_VALUE;
// Duty cycle value of output waveform.
g_pwm_channel_led.ul_duty = INIT_DUTY_VALUE;
pwm_channel_init(PWM, &g_pwm_channel_led);

8.1.2.3 Workflow

1. Define the PWM channel instance, in order to configure channel 0:

   pwm_channel_t g_pwm_channel_led;

2. Enable the module clock for the PWM peripheral:

   pmc_enable_periph_clk(ID_PWM);

3. Disable PWM channel 0:

   pwm_channel_disable(PWM, PWM_CHANNEL_0);

4. Set up the clock for PWM module:

   // Set PWM clock A as PWM_FREQUENCY*PERIOD_VALUE (clock B is not used).
   pwm_clock_t clock_setting = {
       .ul_clka = PWM_FREQUENCY * PERIOD_VALUE,
       .ul_clkb = 0,
       .ul_mck = sysclk_get_cpu_hz()
   };
   pwm_init(PWM, &clock_setting);
5. Initialize the channel instance and configure PWM channel 0, selecting clock A as its source clock and setting the duty cycle at 50%:

```c
const struct pwm_channel LED channel = PWM_CHANNEL_0;

// Period is left-aligned.
const struct pwm_align left = PWM_ALIGN_LEFT;
// Output waveform starts at a low level.
const struct pwm polarity = PWM_LOW;
// Use PWM clock A as source clock.
const struct pwm in cmr CPRE_CLKA = PWM_CMR_CPRE_CLKA;
// Period value of output waveform.
const struct pwm in cmr period = PERIOD_VALUE;
// Duty cycle value of output waveform.
const struct pwm in cmr duty = INIT_DUTY_VALUE;
pwm_channel_init(PWM, &g_pwm_channel_led);
```

### Note
- The period is left-aligned and the output waveform starts at a low level.
- After setting each channel's parameters, the `g_pwm_channel_led` structure can be reused to configure other PWM channels.

8.1.3 Usage Steps

8.1.3.1 Example Code

Add the following code to, for example, the main loop in your application C-file:

```c
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.1.3.2 Workflow

Enable PWM channel 0 and output a square-wave on this channel:

```c
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.1.4 Advanced Use Case

In this use case, the PWM Controller module is configured as follows:

- Output a square-wave on PWM channel 0
- The frequency of the square-wave is 1kHz
- The duty cycle is changed in the PWM's ISR
- Clock A is used as the source clock
- The output wave can be verified on the device's selected output pin
8.1.5 Setup Steps

8.1.5.1 Prerequisites

- Power Management Controller (PMC) driver
- General Purpose I/O (GPIO) Management driver

8.1.5.2 Example Code

Add the following code segments to your application C-file:

```c
pwm_channel_t pwm_channel_instance; *

void PWM_Handler(void)
{
    static uint32_t ul_duty = 0;
    uint32_t ul_status;
    static uint8_t uc_countn = 0;
    static uint8_t uc_flag = 1;

    ul_status = pwm_channel_get_interrupt_status(PWM);
    if ((ul_status & PWM_CHANNEL_0) == PWM_CHANNEL_0) {
        uc_count++;
        if (uc_count == 10) {
            if (uc_flag) {
                ul_duty++;
                if (ul_duty == 100) {
                    uc_flag = 0;
                }
            } else {
                ul_duty--;
                if (ul_duty == 0) {
                    uc_flag = 1;
                }
            }
        }
        uc_count = 0;
        pwm_channel_instance.channel = PWM_CHANNEL_0;
        pwm_channel_update_duty(PWM, &pwm_channel_instance, ul_duty);
    }
}

pmc_enable_periph_clk(ID_PWM);
pwm_channel_disable(PWM, PWM_CHANNEL_0);
pwm_clock_t clock_setting = {
    .ul_clka = 1000 * 100,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);
pwm_channel_instance.ul_prescaler = PWM_CMR_CPRE_CLKA;
pwm_channel_instance.ul_period = 100;
pwm_channel_instance.ul_duty = 0;
pwm_channel_instance.channel = PWM_CHANNEL_0;
pwm_channel_init(PWM, &pwm_channel_instance);
```
8.1.5.3 Workflow

1. Declare the PWM channel instance in order to configure channel 0:
   
   ```c
   pwm_channel_t pwm_channel_instance;
   ```

2. Declare the PWM interrupt handler function in the application:
   
   ```c
   void PWM_Handler(void);
   ```

3. In the function PWM_Handler(), get the PWM interrupt status:
   
   ```c
   ul_status = pwm_channel_get_interrupt_status(PWM);
   ```

4. In the function PWM_Handler(), check if the PWM channel 0 interrupt has occurred:
   
   ```c
   if ((ul_status & PWM_CHANNEL_0) == PWM_CHANNEL_0) {
   ```

5. In the function PWM_Handler(), if the PWM channel 0 interrupt has occurred then update the ul_duty value:
   
   ```c
   uc_count++;
   if (uc_count == 10) {
       if (uc_flag) {
           ul_duty++;
           if (ul_duty >= 100) {
               uc_flag = 0;
           }
       } else {
           ul_duty--;
           if (ul_duty == 0) {
               uc_flag = 1;
           }
       }
   }
   ```

6. In the function PWM_Handler(), if the ul_duty value has been updated then change the square-wave duty:
   
   ```c
   pwm_channel_instance.channel = PWM_CHANNEL_0;
   pwm_channel_update_duty(PWM, &pwm_channel_instance, ul_duty);
   ```

7. Enable the PWM clock:
   
   ```c
   pmc_enable_periph_clk(ID_PWM);
   ```

8. Disable the PWM channel 0:
   
   ```c
   pwm_channel_disable(PWM, PWM_CHANNEL_0);
   ```

9. Setup the clock for the PWM Controller module:
   
   ```c
   * pwm_clock_t clock_setting = {
   ```
.ul_clka = 1000 * 100,
.ul_clkb = 0,
.ul_mck = sysclk_get_cpu_hz();
pwm_init(PWM, &clock_setting);

Note
● Only Clock A is configured (clock B is not used).
● The expected output frequency is 1kHz.

10. Initialize the channel structure and configure PWM channel 0, selecting clock A as its source clock and setting the initial duty cycle as 0%:

    pwm_channel_instance.ul_prescaler = PWM_CMR_CPRE_CLKA;
pwm_channel_instance.ul_period = 100;
pwm_channel_instance.ul_duty = 0;
pwm_channel_instance.channel = PWM_CHANNEL_0;
pwm_channel_init(PWM, &pwm_channel_instance);

Note
● The period is left-aligned and the output waveform starts at a low level.
● The g_pwm_channel_instance structure can be reused to configure other PWM channels, after setting each channel's parameters.

11. Enable the PWM channel 0 interrupt:

    pwm_channel_enable_interrupt(PWM, PWM_CHANNEL_0, 0);

Note
● To enable the PWM interrupt, the NVIC must be configured to enable the PWM Controller interrupt.
● When the channel 0 counter reaches the channel period, the interrupt (counter event) will occur.

8.1.6 Usage Steps

8.1.6.1 Example Code
Add the following code to, for example, the main loop in your application C-file:

    pwm_channel_enable(PWM, PWM_CHANNEL_0);

8.1.6.2 Workflow
Enable PWM channel 0, and output square-wave on this channel:

    pwm_channel_enable(PWM, PWM_CHANNEL_0);

8.2 Pulse Width Modulator Controller - Example controlling an LED

8.2.1 Purpose
This example demonstrates the simple configuration of two PWM channels to generate signals with a variable duty cycle. The brightness of the two LEDs on the evaluation kit will vary repeatedly.
8.2.2 Requirements

This example can be used on any SAM3/4 evaluation kits (except SAM4L).
The two required LEDs need to be connected to PWM output pins, otherwise consider verifying the PWM output signals using an oscilloscope.

8.2.3 Main Files

- pwm.c: Pulse Width Modulator Controller driver
- pwm.h: Pulse Width Modulator Controller driver header file
- pwm_led_example.c: Pulse Width Modulator example application

8.2.4 Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench® for Atmel®. Other compilers may or may not work.

8.2.5 Usage

1. Build the program and download it into the evaluation board.

2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control

3. Start the application.

4. In the terminal window, the following text should appear:

   ```
   -- PWM LED Example --
   -- xxxxxx-xx --
   -- Compiled: xxx xx xxxx xx:xx:xx --
   ```

5. The example code then performs the following:
   - Initialize the system clock and pin setting on the evaluation kit
   - Initialize the PWM clock
   - Configure the PIN_PWM_LED0_CHANNEL
   - Configure the PIN_PWM_LED1_CHANNEL
   - Enable the interrupt on counter event for the PIN_PWM_LED0_CHANNEL and PIN_PWM_LED1_CHANNEL channels
   - Change the duty cycle in the ISR
8.3 Pulse Width Modulator Controller - Example synchronous channel LED control

8.3.1 Purpose
This example demonstrates the simple configuration of two PWM synchronous channels to generate variable duty cycle signals. The duty cycle values are updated automatically by the Peripheral DMA Controller (PDC), which makes two of the on-board LEDs glow repeatedly.

8.3.2 Requirements
This example can be used on any SAM3/4 evaluation kits (except SAM4L). The two required LEDs need to be connected to PWM output pins, otherwise consider verifying the PWM output signals using an oscilloscope.

8.3.3 Main Files
- pwm.c: Pulse Width Modulator Controller driver
- pwm.h: Pulse Width Modulator Controller driver header file
- pwmSync_example.c: Pulse Width Modulator example application

8.3.4 Compilation Information
This software is written for GNU GCC and IAR Embedded Workbench for Atmel. Other compilers may or may not work.

8.3.5 Usage
1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft Windows) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. Start the application.
4. In the terminal window, the following text should appear:

```
-- PWM SYNC Example --
-- xxxxxx-xx --
-- Compiled: xxx xx xxxx xx:xx:xx --

Menu: press a key to change the configuration.

u : Change update period for synchronous channels
d : Change dead time of PWM outputs
  o : Enable/disable output override
```

5. The example code then performs the following:
- Initialize the system clock and the GPIO pins
- Initialize the PWM clock
- Configure the PIN_PWM_LED0_CHANNEL
- Configure the PIN_PWM_LED1_CHANNEL
- Configure the PDC transfer for PWM duty cycle update
- Enable the PDC TX interrupt and PIN_PWM_LED0_CHANNEL
- Update the synchronous period, dead time and override output via UART console
- Restart the PDC transfer in the ISR
Index

E
Enumeration Definitions
  pmc_cmp_unit_t, 31
  pwm_additional_edge_mode_t, 32
  pwm_align_t, 32
  pwm_cmp_interrupt_t, 32
  pwm_counter_event_t, 32
  pwm_fault_id_t, 33
  pwm_level_t, 33
  pwm_pdc_interrupt_t, 33
  pwm_pdc_request_mode_t, 33
  pwm_protect_reg_group_t, 33
  pwm_spread_spectrum_mode_t, 34
  pwm_stepper_motor_pair_t, 34
  pwm_sync_interrupt_t, 34
  pwm_sync_update_mode_t, 34
  _pwm_ch_t, 31

F
Function Definitions
  pwm_channel_disable, 14
  pwm_channel_disable_interrupt, 14
  pwm_channel_enable, 15
  pwm_channel_enable_interrupt, 15
  pwm_channel_get_counter, 15
  pwm_channel_get_interrupt_mask, 16
  pwm_channel_get_interrupt_status, 16
  pwm_channel_get_status, 17
  pwm_channel_init, 17
  pwm_channel_update_additional_edge, 17
  pwm_channel_update_dead_time, 18
  pwm_channel_update_duty, 18
  pwm_channel_update_output, 19
  pwm_channel_update_period, 19
  pwm_channel_update_polarity_mode, 20
  pwm_channel_update_spread, 20
  pwm_cmp_change_setting, 21
  pwm_cmp_disable_interrupt, 21
  pwm_cmp_enable_interrupt, 21
  pwm_cmp_get_period_counter, 22
  pwm_cmp_get_update_counter, 22
  pwm_cmp_init, 23
  pwm_disable_protect, 23
  pwm_enable_protect, 23
  pwm_fault_clear_status, 24
  pwm_fault_get_input_level, 24
  pwm_fault_get_status, 24
  pwm_fault_init, 25
  pwm_get_interrupt_mask, 25
  pwm_getinterrupt_status, 25
  pwm_get_protect_status, 26
  pwm_init, 26
  pwm_pdc_disable_interrupt, 27
  pwm_pdc_enable_interrupt, 27
  pwm_pdc_set_request_mode, 28
  pwm_stepper_motor_init, 28
  pwm_sync_change_period, 28
  pwm_sync_disable_interrupt, 29
  pwm_sync_enable_interrupt, 29
  pwm_sync_get_period_counter, 30
  pwm_sync_init, 30
  pwm_sync_unlock_update, 31

M
Macro Definitions
  PWM_INVALID_ARGUMENT, 14

S
Structure Definitions
  pwm_channel_t, 11
  pwm_clock_t, 12
  pwm_cmp_t, 12
  pwm_fault_t, 13
  pwm_output_t, 13
  pwm_protect_t, 13

T
Type Definitions
  pwm_ch_t, 11
## Document Revision History

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42294A</td>
<td>05/2014</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>